Metadata driven component development

using Beanlet

What is metadata driven component development?

It’s all about POJOs and IoC

- Use Plain Old Java Objects to
  - focus on business logic, and business logic only
  - write Object Oriented code (instead of procedural EJBs)
  - write framework agnostic code
  - simplify testing

Don't call us, we'll call you.

- Add metadata to POJOs to achieve Inversion of Control
  - Framework is instructed how to control POJOs
  - Framework is responsible for
    - Dependency injection
    - Lifecycle management
    - Additional services

Additional services

- Additional services may include
  - Resource management
    - Object pooling
    - Thread management
  - Interceptors
  - Transactions
  - Remoting
    - Web Services
    - Remote Method Invocation

Metadata formats

Annotations

- Pros:
  - Single unit of code
  - Documentation
  - Compile time checks
- Cons:
  - Bound to code (negative impact on code reuse)
  - Requires Java 5 or higher

XML

- Pros:
  - Flexible (no need for recompilation)
  - Applicable to third party code
  - Code has no dependency to underlying framework
- Cons:
  - Verbose (programming in XML is not OK!)

When to use annotations / XML?

- In general, use annotations if
  - the metadata you are applying changes the design of your class
  - the metadata changes the design of code interacting with your class
- Use xml in case of
  - pure configuration, as annotations do not allow dynamic changes (recompilation)

Introduction to Beanlet

- Java Application Container that delivers the flexibility of Spring and the programming model of EJB3
- Configuration based on annotations and XML
- All (!) annotations can be expressed in XML as well. Use XML to define (or override existing) annotations

Beanlet Runtime

- Requires JSE 5 (Tiger) or higher
- Runs stand-alone or embedded inside:
  - Applet containers
  - Servlet containers
  - EJB containers
  - Regular JSE applications

Beanlet and Spring

- Beanlet offers features of Spring:
  - Dependency Injection
  - Good exception reporting
  - Full control through XML configuration

Beanlet and EJB3

- Beanlet brings the power of EJB3 to the client-side:
  - Dependency Injection
  - Lifecycle management
  - Resource management
  - Interceptors
  - Transactions (JTA)
  - Persistence (JPA)

Beyond EJB3

- In addition to EJB3 features, Beanlet provides:
  - Additional component scopes
    - Singleton
    - Web (request & session)
  - Concurrency control (reentrant & non-reentrant)
  - Activation, Execution and Scheduling
  - Management integration (JMX)

Dependency Injection

- Which objects can be injected?
  - Beanlets
  - Spring beans
  - Objects from the JNDI tree
- Supported injection types:
  - Constructor & Setter injection
  - Factory injection
  - Auto wiring

Example Class

```
public class Foo {

   private final Runnable bar;

   public Foo(Runnable bar) {
       this.bar = bar;
   }

   public void someBusinessMethod() {
       bar.run();
   }
}
```

Constructor Injection

```
public class Foo {

   private final Runnable bar;

   @Inject(ref="bar")
   public Foo(Runnable bar) {
       this.bar = bar;
   }

   public void someBusinessMethod() {
       bar.run();
   }
}
```

Constructor Injection

```
<beanlet name="foo" type="com.acme.Foo">
   <inject constructor=“true” ref="bar“/>
</beanlet>
```

Setter Injection (method)

```
@Wiring(BY_NAME)
public class Foo {

   private Runnable bar;

   @Inject
   public void setBar(Runnable bar) {
       this.bar = bar;
   }

   public void someBusinessMethod() {
       bar.run();
   }
}
```

Setter Injection (method)

```
<beanlet name="foo" type="com.acme.Foo">
  <wiring value="BY_NAME"/>
  <inject method="setBar"/>
</beanlet>
```

Setter Injection (field)

```
@Wiring(BY_NAME)
public class Foo {

  @Inject
  private Runnable bar;

  public void someBusinessMethod() {
     bar.run();
  }
}
```

Setter Injection (field)

```
<beanlet name="foo" type="com.acme.Foo">
  <wiring value="BY_NAME"/>
  <inject field="bar"/>
</beanlet>
```

Configuration Injection

```
public class Foo {

  @Inject
  private Collection<Integer> numbers;

  public void someBusinessMethod() {
     for (Integer n : numbers) {
          System.out.println(n);
     }
  }
}
```

Configuration Injection

```
<beanlet name="foo" type="com.acme.Foo">
  <inject field=“numbers“>
     <collection type=“java.util.TreeSet”>
          <value value=“3“/>
          <value value=“2“/>
          <value value=“6“/>
          <value value=“1“/>
     </collection>
  </inject>
</beanlet>
```

Lifecycle Management

```
@Wiring(BY_NAME)
public class Foo {

   @Inject
   private Runnable bar;

   @PostConstruct
   public void init() {
       if (bar == null) throw new NullPointerException();
   }

   public void someBusinessMethod() {
       bar.run();
   }
}
```

Lifecycle Management

```
<beanlet name="foo" type="com.acme.Foo">
  <wiring value="BY_NAME"/>
  <inject field="bar"/>
  <post-construct method="init"/>
</beanlet>
```

Lifecycle Management

```
@Wiring(BY_NAME)
public class Foo {

  @Inject
  private Runnable bar;

  @Schedule(cron="0/5 * * * * *")
  public void someBusinessMethod() {
     bar.run();
  }
}
```

Lifecycle Management

```
@Wiring(BY_NAME)
public class Foo {

  @Inject
  private Runnable bar;

  @Execute
  public void someBusinessMethod() {
     bar.run();
  }
}
```

Lifecycle Management

```
<beanlet name="foo" type="com.acme.Foo">
  <wiring value="BY_NAME"/>
  <inject field="bar"/>
  <execute method="someBusinessMethod"
           loop="true" threads="2"/>
</beanlet>
```

Resource Management

```
public class Bar implements
    Runnable {

  public void run() {
     // Do something…
  }
}
```

Resource Management

```
<beanlet name="foo" type="com.acme.Foo">
  <wiring value="BY_NAME"/>
  <inject field="bar"/>
  <execute method="someBusinessMethod"
           loop="true" threads="2"/>
</beanlet>
<beanlet name="bar" type="com.acme.Bar">
  <stateless reentrant="false"/>
</beanlet>
```

Adaptor

```
public class Bar {

  public void doIt() {
     // Do something…
  }
}
```

Adaptor

```
<beanlet name="foo" type="com.acme.Foo“/>
<beanlet name=“bar" type="com.acme.Bar">
  <proxy type=“java.lang.Runnable”/>
  <operation method=“doIt” name=“run”/>
</beanlet>
```

Interceptors

```
@Wiring(BY_NAME)
public class Foo {

  @Inject
  private Runnable bar;

  @Interceptors(TimeLogger.class)
  public void someBusinessMethod() {
     bar.run();
  }
}
```

Interceptors

```
public class TimeLogger {

  @AroundInvoke
  public void log(InvocationContext ic)
          throws Exception {
     long start = currentTimeMillis();
     ic.proceed();
     long end = currentTimeMillis();
     out.println((end – start));
  }
}
```

Interceptors

```
@Wiring(BY_NAME)
public class Foo {

   @Inject
   private Runnable bar;

   public void someBusinessMethod() {
       bar.run();
   }

   @AroundInvoke
   public void local(InvocationContext ic)
              throws Exception {
       ic.proceed();
   }
}
```

Transactions and JPA

```
@TransactionAttribute(REQUIRED)
public class Foo {

   @PersistenceContext
   private EntityManager em;

   public void someBusinessMethod() {
      FooEntity entity = new FooEntity();
      entity.setName(“beanlet”);
      em.persist(entity);
   }
}
```

Management

```
@Manageable(namingStrategy=IdentityNamingStrategy.class,
            registrationPolicy=REPLACE_EXISTING)
public class Foo {

   private int value;

   public void someBusinessMethod() {
   }

   public int getValue() {
        return value;
   }

   public void setValue(int value) {
        this.value = value;
   }
}
```

Packaging

```
META-INF/beanlet.xml
META-INF/resources.xml
META-INF/transactions.xml
META-INF/beanlet.properties
com/acme/Foo.class
com/acme/Bar.class
```

beanlet.xml

```
<beanlets xmlns="http://beanlet.org/schema/beanlet”
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://beanlet.org/schema/beanlet
        http://beanlet.org/schema/beanlet/beanlet_1_0.xsd">

   <import resource=“resources.xml”/>
   <import resource=“transactions.xml”/>

   <beanlet name=“foo” type=“com.acme.Foo”/>
   <beanlet name=“bar” type=“com.acme.Bar”/>

</beanlet>
```

resources.xml

```
<!-- This is an abstract beanlet. This beanlet is not deployed. It provides
     a template for child beanlets. -->
<beanlet name="abstractDataSource"
         abstract="true"
         type="org.apache.commons.dbcp.BasicDataSource">
   <inject method="setInitialSize" value="1"/>
   <inject method="setMaxActive" value="5"/>
   <inject method="setValidationQuery" optional="true"/>
   <inject method="setDefaultAutoCommit" value="false"/>
   <inject method="setTestOnBorrow" value="true"/>
   <!-- Forces child beanlets to inject the following methods. -->
   <inject method="setDriverClassName"/>
   <inject method="setUrl"/>
   <inject method="setUsername"/>
   <inject method="setPassword"/>
   <pre-destroy method="close"/>
</beanlet>
```

resources.xml

```
<!-- The dataSource beanlet extends the abstractDataSource beanlet. -->
<beanlet name="dataSource" parent="abstractDataSource">
    <inject method="setDriverClassName“
            value="org.apache.derby.jdbc.EmbeddedDriver"/>
    <inject method="setUrl" value="jdbc:derby:Beanlet;create=true"/>
    <!-- The following two methods demonstrate property placeholders. -->
    <inject method="setUsername" value=“${username}"/>
    <inject method="setPassword" value=“${password}"/>
    <!-- The following methods is injected with null explicitly. -->
    <inject method="setValidationQuery" nill="true"/>
</beanlet>
```

beanlet.properties

```
username=beanlet
password=test
```

resources.xml

```
<beanlet name="xaDataSource"
         type="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
   <inject method="setUser" value="${username}"/>
   <inject method="setPassword" value="${password}"/>
   <inject method="setJdbcTestStmt" nill="true"/>
   <inject method="setDataSource">
        <!-- Nested beanlet is injected into setDataSource method. -->
        <beanlet type="org.enhydra.jdbc.standard.StandardXADataSource">
            <inject method="setTransactionManager"/>
            <inject method="setDriverName"
                    value="org.apache.derby.jdbc.EmbeddedDriver"/>
            <inject method="setUrl"
                    value="jdbc:derby:Beanlet;create=true"/>
           <inject method="setUser" value="${username}"/>
            <inject method="setPassword" value="${password}"/>
        </beanlet>
   </inject>
</beanlet>
```

transactions.xml

```
<beanlet name="org.objectweb.jotm.Jotm“
         type="org.objectweb.jotm.Jotm">
   <inject constructor="true" index="0" value="true"/>
   <inject constructor="true" index="1" value="false"/>
   <pre-destroy method="stop"/>
</beanlet>
<beanlet type="javax.transaction.TransactionManager"
         factory="org.objectweb.jotm.Jotm"
         factory-method="getTransactionManager"/>
<beanlet type="javax.transaction.UserTransaction"
         factory="org.objectweb.jotm.Jotm"
         factory-method="getUserTransaction"/>
```